PostgreSQL: Instalación y primeros pasos

Borja Lopez Río

wu@e-shell.org

PostgreSQL es, según muchos, el sistema gestor de bases de datos open source más avanzado de los existentes, llegando incluso a rivalizar en prestaciones con sistemas comerciales como Oracle. En este documento no voy a entrar en polémicas ni comparativas con otros sistemas, si no que voy a explicar como poner a funcionar un sistema como PostgreSQL de una forma básica, desde su instalación a las primeras configuraciones. En siguientes capitulos veremos configuraciones más avanzadas, opciones de optimización, etc...

Borja López, administrador de sistemas de 24 años, comenzo su andadura en el mundo unix a los 17, tomando contacto con estaciones sun con SunOS, evolucionando luego a sistemas con Linux y BSD, con los que lleva trabajando mas de 4 años. En este tiempo a colaborado con proyectos como eSlack, HispaBSD y algunos proyectos de documentacion además de ofrecer sus servicios a todo tipo de empresas relacionadas con internet y el mundo Unix.

Contacto: <`wu@e-shell.org`>

Tabla de contenidos

1. Instalación

Vamos a ver en este apartado como instalar PostgreSQL tanto en FreeBSD como OpenBSD utilizando el sistema de ports de cada uno, asi como la instalación compilando los sources por nuestra cuenta

1.1. FreeBSD

La mejor forma de instalar PostgreSQL en FreeBSD es mediante el sistema de ports, aunque siempre podremos bajarnos el paquete precompilado de ftp://ftp.freebsd.org e instalarlo con *pkg_add*.

Para instalar PostgreSQL desde los ports de FreeBSD, vamos al directorio `/usr/ports/databases/postgresql7`. En el directorio `databases` de los ports hay más directorios que hacen referencia a pgsql como `postgresql72`, que es utilizado para instalar la ultima version de la rama 7.2.x de pgsql.

Una vez en el directorio `postgresql7`, editamos el Makefile para añadir unas cosillas que, en mi opinion, le faltan a la instalación de PostgreSQL de FreeBSD, como son el soporte nativo de Perl y Python. Para añadir estas funcionalidades, solo tenemos que añadir estas lineas al Makefile (evidentemente para añadir estas dos funcionalidades deberán de estar instalados tanto Python como Perl en nuestro sistema, de otra forma nos daría un error, ya que este patch, por así llamarlo, no comprueba esas depencias. Estoy trabajando en uno que si lo haga XD) despues de las lineas en las que establece las variables *CONFIGURE_ARGS* y *CONFIGURE_ENV*:

```
.if defined(WITH_PYTHON)
CONFIGURE_ARGS+=--with-python
.endif

.if defined(WITH_PERL)
CONFIGURE_ARGS+=--with-perl
.endif
```

De esta forma añadimos dos opciones mas al make a la hora de compilar el paquete, ya me he puesto en contacto con el maintainer del port para comunicarle que añadiese esas opciones al Makefile, solo queda esperar que me haga caso XD). Ahora es el momento de compilarlo, para ello, simplemente ejecutamos dentro del directorio `/usr/ports/databases/postgresql7`:

```
make -DWITH_PYTHON -DWITH_PERL -DWITH_LIBC_R install
```

Y si todo va bien ya tendremos el paquete de PostgreSQL 7.3.X instalado, ahora limpiamos el directorio de trabajo y borramos los ficheros que fueron necesarios para la compilación e instalacion de PostgreSQL.

```
make -DWITH_PYTHON -DWITH_PERL -DWITH_LIBC_R clean
make -DWITH_PYTHON -DWITH_PERL -DWITH_LIBC_R distclean
```

1.2. OpenBSD

En el caso de OpenBSD también podemos instalar PostgreSQL desde ports, pero nos encontraremos con una pequeña sorpresa, es practicamente imposible que tengan disponible la última version de

PostgreSQL. En el momento de escribir este doc, la última versión de PostgreSQL -stable es la 7.3.4, mientras que los ports de OpenBSD tienen la 7.3.2. La solución para tener la última version de PostgreSQL en OpenBSD es seguir los pasos del siguiente apartado, donde veremos como compilar de 0 PostgreSQL.

Para instalar desde ports, nos encontramos con el mismo problema que en FreeBSD, no hay soporte nativo para los modulos de Perl ni Python, por lo que tendremos que ir al directorio `/usr/ports/databases/postgresql` y editar el Makefile para añadir ese soporte. A diferencia de como lo hicimos en el caso de FreeBSD, añadiendo dos nuevas opciones al make, aqui añadimos los argumentos al configure directamente. Estoy trabajando de todas formas en un nuevo Makefile para el port de PostgreSQL de OpenBSD, con los dos nuevos *FLAVORS* definidos, pronto mas noticias en el cvs de OpenBSD (eso espero XD).

```
CONFIGURE_ARGS= --disable-rpath \
                --enable-integer-datetimes \
                --includedir="${PREFIX}/include/postgresql" \
                --datadir="${PREFIX}/share/postgresql" \
                --docdir="${PREFIX}/share/doc/postgresql" \
                --with-python --with-perl$
```

Una vez twekeado el Makefile, y añadidas las opciones *-with-python* y *-with-perl* podemos instalar PostgreSQL facilmente, simplemente:

1.3. Instalando desde sources

1.3.1. Obteniendo los sources

Lo primero es lo primero, y en este caso hemos de bajarnos los sources de PostgreSQL.org, para ello podemos ir a la pagina web, http://www.postgresql.org y seleccionar el mirror que más nos guste, o ir directamente al mirror belga (belnet, muy rápido): ftp://ftp.be.postgresql.org/postgresql

Y nos bajamos los sources de la ultima version, que en el momento de escribir este doc, es la 7.3.4: ftp://ftp.be.postgresql.org/postgresql/source/v7.3.4

Los bajamos a un directorio donde vamos a realizar todo el trabajo, como por ejemplo `/root/pgsql`.

```
[Silence] ~/pgsql# ls -l
total 2
drwxr-xr-x  3 root  wheel  512 Sep 20 12:43 ftp.be.postgresql.org
[Silence] ~/pgsql#
```

1.3.2. Compilando

El proceso de compilación de PostgreSQL es sencillo, vamos a ver las opciones más interesantes a la hora de configurar el software antes de compilarlo e instalarlo en el sistema. En estas pruebas voy a instalar el PostgreSQL en `/usr/local/pgsql`, de forma que quede instalado un poco a parte del sistema base, pero esto depende del gusto de cada uno.

Lo primero es descomprimir los sources, lo que haremos con el comando **tar**:

```
[Silence] ~/pgsql# tar -zxvvf ftp.be.postgresql.org/postgresql/source/v7.3.4/postgresql-7.3.4.tar.gz
```

Con descomprimir ese `postgresql-X.X.X.tar.gz` llega, los otros ficheros .tar.gz son las diferentes partes del PostgreSQL en ficheros separados, pero en este paquetito tenemos todo lo que necesitamos.

Una vez descomprimido, podemos explorar un poco el contenido de los sources:

```
[Silence] ~/pgsql/postgresql-7.3.4# ls -l
total 770
-rw-r--r--   1 pgsql  wheel    1196 Jun 20  2002 COPYRIGHT
-rw-r--r--   1 pgsql  wheel    3435 Oct 21  2002 GNUmakefile.in
-rw-r--r--   1 pgsql  wheel  189570 Jul 24 02:44 HISTORY
-rw-r--r--   1 pgsql  wheel   46083 Jul 23 06:09 INSTALL
-rw-r--r--   1 pgsql  wheel    1432 Feb 10  2001 Makefile
-rw-r--r--   1 pgsql  wheel    1376 Nov 11  2002 README
-rw-r--r--   1 pgsql  wheel     449 Sep  5  2002 aclocal.m4
drwxr-xr-x   2 pgsql  wheel     512 Jul 25 00:51 config
-rwxr-xr-x   1 pgsql  wheel  468439 Jul 23 06:09 configure
-rw-r--r--   1 pgsql  wheel   36758 Jul 23 06:09 configure.in
drwxr-xr-x  48 pgsql  wheel    1024 Jul 25 00:51 contrib
drwxr-xr-x   4 pgsql  wheel    1024 Jul 25 00:51 doc
-rw-r--r--   1 pgsql  wheel     687 Oct 24  2002 register.txt
drwxr-xr-x  16 pgsql  wheel     512 Jul 25 00:54 src
[Silence] ~/pgsql/postgresql-7.3.4#
```

Son especialmente interesantes los ficheros `README` e `INSTALL`. El primero es una introducción a los contenidos de este directorio en el que estan los sources de PostgreSQL, el segundo son las instrucciones específicas de como configurar, compilar e instalar PostgreSQL (este es más que aconsejable para leer antes de ponerse a hacer nada más).

Bien, una vez leido el `INSTALL`, y comprendidas las opciones del PostgreSQL, *lo que necesitamos y lo que no* (aquí es donde radica el poder de compilar por ti mismo este tipo de cosas, ya que puedes optimizarlo al máximo para tus necesidades), pasamos a la configuración de los sources previa a la compilación de los mismos. Este proceso es típico de software en sistemas unix-like, basta con ejecutar el configure de turno con las opciones adecuadas... .

```
[Silence] ~/pgsql# /configure --prefix=/usr/local/pgsql --with-openssl=usr \
 --enable-syslog --with-python --with-perl
```

En nuestro caso le he pasado las siguientes opciones:

-prefix=/usr/local/pgsql

Aquí le especifico el path donde será instalado el PostgreSQL al hacer el make install.

-with-perl -with-python

Soporte empotrado/embebido de Perl y Python, si activamos estas opciones, se compilaran e instalar modulos para acceder desde ambos lenguajes a la base de datos. En el caso de Python, que es el que he usado, instala el modulo pg, mediante el cual puedes trabajar contra el sistema gestor de bases de datos desde tus programas en Python.

-with-openssl=usr

Soporte SSL, para poder realizar conexiones encryptadas via *TLS*, con `usr` le estoy diciendo que pille las libs openssl instaladas en `/usr` (en el sistema base), aquí se le podrían especificar otros lugares como `/usr/local/openssl` para instalaciones alternativas que pudiesemos tener de openssl.

-enable-syslog

Soporte para que loguee al syslog del sistema los posibles fallos, mensajes, etc.

Una vez configurados los sources, es el momento de compilarlos, para lo cual ejecutamos:

```
[Silence] ~# gmake
```

Tardará un rato, dependiendo del procesador de la maquina, en compilar tanto la parte de servidor como las librerías y los clientes que trae (módulos para interactuar con perl/python, el comando psql, pg_dump, etc...). En cuanto acabe:

```
[Silence] ~# gmake install
[Silence] ~# gmake install-all-headers
```

Este último para que instale los includes por si vamos a desarrollar algo contra las librerías de PostgreSQL.

Si todo ha ido bien hasta ahora, deberiamos de tener todo lo necesario para correr PostgreSQL en `/ust/local/pgsql`.

2. Primeras configuraciones

Una vez instalado PostgreSQL, ahora tendremos que configurar un par de cosas para poder ponerlo a funcionar. En este apartado veremos como iniciar el directorio database cluster, que es donde el gestor de bases de datos guardará la información, asi como las formas de lanzar el daemon de PostgreSQL y los primeros pasos para crear usuarios que utilicen el sistema y como darles acceso.

A lo largo del apartado se harán referencias a comandos ejecutados desde shell que requieren como parametro determinados *paths* o *rutas* referentes a la instalacion del PostgreSQL, vamos a tomar como valor siempre `/usr/local/pgsql`, por ser el directorio usado tanto por la instalacion desde ports de FreeBSD como la instalación hecha desde sources.

2.1. Usuario pgsql

Lo primero que necesitamos es un usuario que se va a encargar de lanzar el daemon de PostgreSQL, y será el dueño de los ficheros del sistema gestor de bases de datos. En el caso de FreeBSD y OpenBSD, instalando desde ports, no es necesario este paso, ya que el propio port se encarga de hacerlo por nosotros, pero si hemos instalado por sources tendremos que hacerlo nosotros a mano.

Se le puede llamar como uno quiera, yo suelo llamarlo pgsql. Antes de crear al usuario, tenemos que crear el grupo:

```
[Silence] ~# groupadd pgsql
```

Y luego el usuario al fin.

```
[Silence] ~# useradd -c "PostgreSQL Admin User" -d /home/pgsql -g pgsql -m -p XXXXXXXXX -s /bin/tcsh pgsql
```

Con esto estamos creando el usuario pgsql, con home en `/home/pgsql`, pertenecietne al grupo pgsql, con shell tcsh (depende de que BSD sea, en OpenBSD es aconsejable `/bin/ksh`, mientras que en

FreeBSD `/bin/tcsh`, aunque siempre podemos poner la que más nos guste) y con password XXXXXXXXX.

Verificamos que funciona el usuario y que todo esta bien:

```
[Silence] ~# su - pgsql
$ pwd
/home/pgsql
$ id
uid=518(pgsql) gid=512(pgsql) groups=512(pgsql)
```

Este va a ser el encargado, por lo menos al principio, de administrar/lanzar/detener el sistema gestor de bases de datos.

2.2. Preparando el *database cluster*

Para que PostgreSQL pueda empezar a funcionar, tenemos que habilitar primero un lugar donde el sistema almacenará la información de las bases de datos. Para hacer esto, tenemos el comando **initdb**, que nos permite crear ese espacio en el lugar del sistema de ficheros que queramos. Para generar ese espacio en el directorio `/usr/local/pgsql/data` hemos de llamar a **initdb** como el usuario pgsql:

```
$ initdb --locale=es_ES -D /usr/local/pgsql/var/data
The files belonging to this database system will be owned by user "pgsql".
This user must also own the server process.


The database cluster will be initialized with locale es_ES.
This locale setting will prevent the use of indexes for pattern matching
operations.  If that is a concern, rerun initdb with the collation order
set to "C".  For more information see the Administrator's Guide.

creating directory /usr/local/pgsql/data... ok
creating directory /usr/local/pgsql/data/base... ok
creating directory /usr/local/pgsql/data/global... ok
creating directory /usr/local/pgsql/data/pg_xlog... ok
creating directory /usr/local/pgsql/data/pg_clog... ok
creating template1 database in /usr/local/pgsql/data/base/1... ok
creating configuration files... ok
initializing pg_shadow... ok
enabling unlimited row size for system tables... ok
initializing pg_depend... ok
creating system views... ok
loading pg_description... ok
creating conversions... ok
setting privileges on built-in objects... ok
vacuuming database template1... ok
copying template1 to template0... ok

Success. You can now start the database server using:

    /usr/local/pgsql/bin/postmaster -D /usr/local/pgsql/data
```

```
or
    /usr/local/pgsql/bin/pg_ctl -D /usr/local/pgsql/data -l logfile start

$
```

La opcion *-locale=es_ES* sirve para que todas las bases de datos que se creen en ese entorno adopten esas locales. Esto es *MUY MUY importante*, ya que *sin la local adecuada al hacer ordenaciones en los selects, nos devuelve ordenaciones erroneas como por ejemplo, a la hora de ordenar caracteres, la ordenacion primero de minusculas y luego mayusculas, o primero sin tilde y luego con tilde*. Como lo más probable es que nuestros usuarios inserten datos con caracteres tipicos españoles como á,é,í,ó,ú... es necesario que las bases de datos se creen con locale a español. Una nota a tener en cuenta es que si las locales de nuestro sistema ya están especificadas en español, no es necesaria esta opción, ya que por defecto el **initdb** pilla la locale que está usando nuestro sistema.

2.3. Lanzando el daemon

A la hora de lanzar el daemon de PostgreSQL tenemos dos opciones diferenciadas, *postmaster* y *pg_ctl*. El primero es el daemon en si, que podemos lanzar llamandolo directamente y pasandole las opciones pertinentes, mientras que el segundo es una interfaz del primero, es decir, llama a postmaster con unas opciones determinadas, más otras que le pasemos nosotros.

En este apartado vamos a ver ambas opciones, como lanzar el daemon y como automatizar el proceso para que se relance solo el daemon en caso de encontrarnos con un fallo de hardware o de energia y se reincie el servidor.

2.3.1. postmaster

Postmaster es el daemon en si, por lo que al lanzarlo directamente, hará un fork y se pondra en background a escuchar las peticiones de los clientes y usuarios que se conecten a las bases de datos; tiene multitud de opciones que podemos consultar en la página **man** de postmaster.

Para lanzar simplemente postmaster, podemos hacerlo de la siguiente manera:

```
$ postmaster -i -D /usr/local/pgsql/data $
```

Al hacer esto, veremos que el daemon no se pone en background, si no que se queda mostrando por stdout un log (el log del sistema), esto es normal al no invocarlo como daemon, que veremos más adelante. Las opciones que se le pasan como parametros son:

-I

Necesaria para que el PostgreSQL escuche peticiones tcp en el puerto 5432, sin esta opción activada, el sistema de bases de datos solo aceptaría conexiones locales, dentro del mismo host.

-D

Este parametro le dice al postmaster que lo que sigue es el directorio que antes creamos (el *database cluster*) y donde está ese entorno donde PostgreSQL realizará las operaciones con la información. Esta es una manera un poco simplona de lanzar el PostgreSQL, pero para hacer las primeras pruebas está bien.

Una vez lanzado, lo primero es comprobar si funciona y si ha echo el bind al puerto 5432 correctamente. Lo primero es sencillo, basta con usar el comando **psql** para que liste las bases de datos que existen, para lo que ejecutamos (siempre como usuario *pgsql*):

```
$ psql -l
        List of databases
   Name    |  Owner   | Encoding
-----------+----------+-----------
 template0 | pgsql    | SQL_ASCII
 template1 | pgsql    | SQL_ASCII
(2 rows) $
```

Si no da ningun error, y nos lista las bases de datos *template0* y *template1*, todo esta perfecto para trabajar, ya solo nos queda comprobar las conexiones a traves del puerto *5432*, lo cual podemos verificar a través de:

```
[Silence] ~# netstat -an | grep LISTEN | grep 5432
tcp4       0      0  *.5432                 *.*                    LISTEN
[Silence] ~#
```

Ahora vamos a ver un pequeño script que nos va a permitir lanzar/parar/reiniciar el servidor de forma rapida y ejecutandolo como cualquier usuario, script que podremos utilizar para automatizar el trabajo de arranque/parada del PostgreSQL cuando se reinicie el servidor:

```
#!/bin/sh
#
# Script de control de postmaster, con este script controlamos
# el arranque y parada del daemon de bases de datos PostgreSQL
#
# Wed Sep  3 16:37:17 CEST 2003 - Borja Lopez -
#
# Basado en el script de control de PostgreSQL creado por mi para Slackware Linux:
# http://wulabs.e-shell.org

# Vamos a definir algunas variables
# para que el script sea portable

PGDATA="/usr/local/pgsql/data"                 # Directorio de trabajo del PostgreSQL
PGLOGFILE="/usr/local/pgsql/pgsql_log"         # Fichero a donde enviaremos el log del servidor
PIDFILE="$PGDATA/postmaster.pid"               # Fichero donde guardo el PID del postmaster
PGPID=`head -1 $PIDFILE 2> /dev/null`          # El id del proceso postmaster, para luego poder pararlo
PGADMINUSER="pgsql"                            # El usuario que lanzara el postmaster

# Comandos que vamos a necesitar, de esta forma hacemos
# que el script sea lo mas portable posible.

POSTMASTER=`which postmaster`
KILL=`which kill`
CPCMD=`which cp`
LNCMD=`which ln`
LSCMD=`which ls`
MKDIRCMD=`which mkdir`
CHOWNCMD=`which chown`
```

```
TARCMD=`which tar`
RMCMD=`which rm`
SUCMD=`which su`

# Funcion que arranca el servidor PostgreSQL, lanzando el postmaster,
# mandandolo a bg y logueando en el fichero establecido por la variable
# $PGLOGFILE.

pgsql_start()
  {
   if [ -x $POSTMASTER ]
   then
    if [ -d $PGDATA ]
     then
      echo "Lanzando daemon PostgreSQL          [ $POSTMASTER -i -D $PGDATA]"
      $SUCMD $PGADMINUSER -c "$POSTMASTER -i -D $PGDATA >> $PGLOGFILE 2&>1 &"
     else
      echo "$PGDATA no es un directorio base valido para PostgreSQL !!"
    fi
   else
    echo "[ERROR] - No puedo lanzar el daemon por que no se donde esta $POSTMASTER !!"
  fi
  }

# Funcion que detiene el servidor de PostgreSQL, basicamente le lanza un SIGTERM, que es la forma
# mas limpia de detener el daemon.

pgsql_stop()
 {
  if [ -x $KILL ]
   then
    echo "Paralo pol! Matando el daemon del PostgreSQL           [ $KILL -15 $PGPID ]"
    $SUCMD $PGADMINUSER -c "$KILL -INT $PGPID"
   else
    echo "Me da que el comando kill no ta ande tu piensas.... [ $KILL ]"
  fi
 }

# Funcion para reinciar el servidor de PostgreSQL, le mandamos un SIGHUP
# al PID del server, para que se recargue de una forma rapida

pgsql_restart()
 {
  if [ -x $KILL ]
   then
    echo "Reiniciando el daemon del PostgreSQL           [ $KILL -1 $PGPID ]"
    echo "Recargando configuracion pg_hba.conf                                "
    $SUCMD $PGADMINUSER -c "$KILL -1 $PGPID"
   else
    echo "Me da que el comando kill no ta ande tu piensas.... [ $KILL ]"
  fi
 }
```

```
# Ahora el case, validando que es lo que pasa el usuario:

case "$1" in
        'start')
                pgsql_start
                ;;
        'stop')
                pgsql_stop
                ;;
        'restart')
                pgsql_restart
                ;;
        *)
                echo "Use - $0 start|stop|restart"
esac
```

Como vemos el script es muy sencillo, simplemente establece una serie de rutas y de binarios que luego utilizaremos, de forma que sea mas o menos portable, los binarios los pilla automaticamente y los valores que podriamos tener que cambiar son 3 o 4 al principio del script, sin que luego tengamos que rebuscar en el codigo.

Dispone de dos opciones básicas, *start* y *stop*, que lanzan o detienen el daemon, y de una tercera, *restart*, que sirve para que el PostgreSQL recargue algunas configuraciones. En principio al postmaster tan solo se le pasan dos opciones, las dos que vimos anteriormente.

2.3.2. pg_ctl

El comando **pg_ctl** es un *wrapper*, que simplemente llama a postmaster con unas opciones determinadas. sus funcionalidades son parecidas al script que he creado para llamar a postmaster, nos da opción a lanzar/parar/relanzar el servidor. Podemos sacar mas información en la página man de pg_ctl.

Basicamente las opciones de pg_ctl son:

pg_ctl start

Se encarga de lanzar el daemon, ponerlo en background y redireccion la salida de stdout y stderr a un fichero de log si se le pasa la opción -l.

pg_ctl stop

Se encarga de para el daemon, dependiendo de la opcion que se le pase (-ms, -mf o -mi) lanzara un tipo de señal al server (espera por los usuarios y SIGTERM, SIGTERM y SIGKILL respectivamente).

pg_ctl restart

Esta opción llama a stop y luego a start sin más.

pg_ctl reload

Similar al restart de nuestro script, le envia una señal SIGHUP al server de forma que relea los ficheros de configuracion.

pg_ctl status

Comprueba si el postmaster esta corriendo y nos muestra tanto el pid, como los parametros con los que se lanzó. Personalmente es una opción que no suelo utilizar, no me parece tan limpia y clara como llamar al postmaster por mi cuenta y lanzar las señales (SIG*) que quiera al proceso del postmaster. Bajo mi punto de vista se pierde algo de control sobre el servidor.

2.4. Usuarios

A la hora de utilizar el sistema gestor de bases de datos, lo mejor es crear un usuario que controlará tanto la creación de las bases de datos como de usuarios, que sea diferente al administrador como el que corre el PostgreSQL. En los siguientes subapartados veremos como se crean usuarios en postgresql y como se le asignan permisos para que acceda o no a determinadas bases de datos.

2.4.1. Creacion, borrado, manejo

Como he dicho antes, lo mejor es crear un usuario que no va a tener nada que ver con el usuario pgsql como el que corre el daemon de PostgreSQL, para que luego sea este el que crea a los demás usuarios, por lo que vamos a ver como se crearía este usuario y luego veremos como creamos algún otro usuario, como lo borramos o le cambiamos parametros de configuración.

Primero nos conectamos a la base de datos template1 como el usuario pgsql:

```
$ whoami
pgsql
$ psql template1
Welcome to psql 7.3.4, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

template1=#
```

psql es la utilidad de gestion de base de datos en linea de comandos de PostgreSQL, es aqui donde vamos a ejecutar las consultas SQL que queramos, etc..., y creamos al usuario:

```
template1=# create user pgadmin with encrypted password 'loveBSD' createdb createuser;
CREATE USER
template1=#
```

A la sentencia *create user* le pasamos algunas opciones:

encrypted

Hace que el password se guarde en la tabla de usuarios de PostgreSQL en formato md5, en lugar de en texto plano.

createdb

Con esta opción, indicamos que el usuario va a poder crear bases de datos. Lo ideal es que tan sólo este usuario pueda crear bases de datos, una vez creadas estas, se les puede cambiar el dueño, asi como asignar permisos a quien queramos, lo que es más que aconsejable.

createuser

Aqui indicamos que este usuario va a tener permisos para crear más usuarios.

El manejo de usuarios en PostgreSQL no se diferencia mucho del manejo de cualquier otro objeto, ya sean bases de datos, tablas, vistas, etc... .

Todo se basa en tres sentencias básicas *create, drop y alter*, podemos encontrar mas ayuda acerca de estas sentencias en la documentacion oficial de PostgreSQL de http://www.postgresql.org, y en el propio psql:

```
template1=# \h create user
Command:     CREATE USER
Description: define a new database user account
Syntax:
CREATE USER username [ [ WITH ] option [ ... ] ]

where option can be:

      SYSID uid
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | CREATEDB | NOCREATEDB
    | CREATEUSER | NOCREATEUSER
    | IN GROUP groupname [, ...]
    | VALID UNTIL 'abstime'
template1=# \h drop user
Command:     DROP USER
Description: remove a database user account
Syntax:
DROP USER name
template1=# \h alter user
Command:     ALTER USER
Description: change a database user account
Syntax:
ALTER USER username [ [ WITH ] option [ ... ] ]

where option can be:

      [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | CREATEDB | NOCREATEDB
    | CREATEUSER | NOCREATEUSER
    | VALID UNTIL 'abstime'

ALTER USER username SET variable { TO | = } { value | DEFAULT }
ALTER USER username RESET variable

Por ejemplo, para crear un usuario BSD, luego cambiarle el password y finalmente borrarlo, lo hariamos sencillamente as\x{00ED}:

template1=# create user BSD;
CREATE USER
```

```
template1=# alter user BSD with encrypted password 'LoveBSD';
ALTER USER
template1=# drop user BSD;
DROP USER
```

2.4.2. Permisos

Una vez creado el usuario de control de las bases de datos y usuarios, el siguiente paso es echarle un vistazo al fichero de control de acceso de Postgresql, el `pg_hba.conf`, donde establecemos quién se puede conectar y a qué.

El fichero `pg_hba.conf` tiene el siguiente formato, como podemos ver si le echamos un ojo al propio fichero:

```
# local    DATABASE  USER  METHOD  [OPTION]
# host     DATABASE  USER  IP-ADDRESS  IP-MASK  METHOD  [OPTION]
# hostssl  DATABASE  USER  IP-ADDRESS  IP-MASK  METHOD  [OPTION]
```

Aquí establecemos por columnas: tipo de conexion, nombre de db, nombre de usuario que se conectara a la db y metodo de conexion.

En el caso de que el tipo de conexion sea host o hostssl podemos establecer *desde que ips o desde que subred de ips se podrá conectar el usuario.*

En el `tipo de conexiones` podemos establecer 3 valores:

local

Conexiones de usuarios dentro del propio host, por ejemplo un usuario conectado por ssh, que ejecuta un psql a una base de datos.

host

Conexion tcp a traves del puerto 5432, aqui le podemos especificar desde que ip o subred se puede conectar el usuario, por ejemplo un usuario desde otro host haciendo *psql -h host_db nombredb*, o un usuario conectandos *desde un servidor web con php*.

hostssl

Igual que el anterior, solo que las conexiones van en un tunel ssl. PostgreSQL, correctamente configurado, soporta conexiones seguras a través del protocolo TLS.

El *método de conexión* se refiere a como se autentica el usuario en el gestot de bases de datos. Los posibles valores pueden ser:

trust

Con este metodo confiamos plenamente en el usuario y ni siquiera le autenticamos, con que se identifique como ese nombre de usuario, se podrá conectar.

password

Metodo usado si hemos establecido un password en el momento de crear al usuario con create user, pero sin usar la opcion encrypted.

md5

Esta si hemos utilizado la opcion encrypted.

ident

Autenticando al usuario por su ident, usuario@ ...

Estos son los mas utilizados, pero ademas tenemos las opciones de *kerberos (krb4 y krb5)*, en el caso de linux tenemos la opcion de los *plugable autentication modules (PAM)*, la opcion *crypt* (en desuso) y la opcion *reject que es como banearlo del servidor.*

En un principio, lo ideal es no tener *ninguna linea trust*, y los usuarios deberian de tener todos la opcion *md5*, de forma que nos encontrariamos con un `pg_hba.conf` en nuestro caso, de la siguiente forma:

```
# TYPE  DATABASE    USER        IP-ADDRESS        IP-MASK           METHOD

local   all         pgadmin                                          md5
host    all         pgadmin     127.0.0.1         255.255.255.255    md5
```

En principio tenemos este pequeño `pg_hba.conf`, en el que nuestro administrador de bases de datos y usuarios tendria acceso a todas las bases de datos tanto conectando desde el propio host (a traves del sock correspondiente) como conectandose tb desde el mismo host por tcp.

Ahora, cada vez que se cree una nueva base de datos, ha de añadirse una linea a este fichero que le de permisos de conexión tan solo al usuario que va a utilizarla.

Cada vez que se haga un cambio a este fichero, tendremos que relanzar el servidor (**rc.pgsql restart** en nuestro caso).

3. Conclusión

Al acabar este articulo deberíamos de tener instalado y configurado un servidor de bases de datos PostgreSQL bajo OpenBSD o FreeBSD. Por el momento es un servidor limitado, que simplemente nos servirá para ir viendo como funciona este sistema gestor de bases de datos, hacernos familiar con él, poder jugar un poco con SQL y con los comandos propios de PostgreSQL, etc...

En el siguiente articulo veremos un poco más a fondo la estructura interna de PostgreSQL, asi como los parametros de configuración que nos permitirán optimizarlo a tope para nuestras necesidades.

Si tienes cualquier sugerencia sobre PostgreSQL en si, sobre el tema de este articulo o sobre como está escrito, puedes contactar conmigo por correo electronico: <`wu@e-shell.org`> o en el IRC, en la red de freenode, irc.eu.freenode.net, en el canal #BSDes.